Formal modelling and verification of interlocking systems featuring sequential release

Vu, Linh Hong; Haxthausen, Anne Elisabeth; Peleska, Jan

Published in:
Science of Computer Programming

Link to article, DOI:
10.1016/j.scico.2016.05.010

Publication date:
2017

Document Version
Peer reviewed version

Formal Modelling and Verification of Interlocking Systems Featuring Sequential Release

Linh H. Vu[a,1], Anne E. Haxthausen[a,1], Jan Peleska[b,2]

[a] *DTU Compute, Technical University of Denmark, Kongens Lyngby, Denmark.*
[b] *Department of Mathematics and Computer Science, University of Bremen, Bremen, Germany.*

Abstract

In this article, we present a method and an associated toolchain for the formal verification of the new Danish railway interlocking systems that are compatible with the European Train Control System (ETCS) Level 2. We have made a generic and reconfigurable model of the system behaviour and generic safety properties. This model accommodates *sequential release* – a feature in the new Danish interlocking systems. To verify the safety of an interlocking system, first a domain-specific description of interlocking configuration data is constructed and validated. Then the generic model and safety properties are automatically instantiated with the well-formed description of interlocking configuration data. This instantiation produces a model instance in the form of a Kripke structure, and concrete safety properties expressed as invariants. Finally, using a combination of SMT based bounded model checking (BMC) and inductive reasoning, it is verified that the generated model instance satisfies the generated safety properties. Using this method, we are able to verify the safety properties for model instances corresponding to railway networks of industrial size. Experiments show that BMC is also efficient for finding bugs in the railway interlocking designs. Additionally, benchmarking results comparing the performance of our approach with alternative verification techniques on the interlocking models are presented.

Keywords: railway interlocking systems, sequential release, formal verification, bounded model checking, inductive reasoning, RobustRailS, openETCS, safety-critical systems

1. Introduction

An interlocking system is responsible for guiding trains safely through a given railway network. It is a vital part of any railway signalling system and has the highest safety integrity level (SIL4) according to the CENELEC 50128 standard [31]. Conventionally, the development and verification process of interlocking systems is informal and mostly manual, and hence time-consuming, costly, and error-prone. Automated verification of interlocking systems is therefore an active research topic, investigated by several research groups, see e.g., [20, 17, 38, 25, 19, 24]. As part of the RobustRailS research project,[3] our work aims at establishing a holistic method supporting the verification of such systems. The method should be formal and facilitate automation in order to provide a better verification process compared to the conventional one. In Denmark, in the period of 2009–2021, new interlocking systems that are compatible with the standardised European Train Control System (ETCS) Level 2 [13] will be deployed in the entire country within the context of the Danish Signalling Programme.[4] In the context of the RobustRailS project accompanying the signalling programme on a scientific level, the proposed method will be applied to these new systems.

Email addresses: lvho@dtu.dk (Linh H. Vu), aeha@dtu.dk (Anne E. Haxthausen), jp@cs.uni-bremen.de (Jan Peleska)
[1] The authors' research has been funded by the RobustRailS project granted by Innovation Fund Denmark.
[2] The author's research has been partially funded by ITEA2 project openETCS under grant agreement 11025.
[3] http://robustrails.man.dtu.dk
[4] http://www.bane.dk/signalprogrammet

The main contributions presented in this article are the following. (1) We present a formal model of the behaviour of ETCS Level 2 compatible interlocking systems. (2) The model accommodates sequential release: this is a method for incrementally releasing route portions that have been traversed by the associated train, with the objective to increase the level of concurrency in route allocation and, consequently, the train throughput in the railway network. On the other hand, this feature poses extra challenges to the verification tasks as it makes the models more complex. (3) The state space encodings allow for high-level safety properties and state transition relations to be processed in a highly efficient manner by Satisfiability Modulo Theories (SMT) solvers supporting bit vector and integer arithmetic. (4) A verification technique combining induction with bounded model checking (BMC) using novel SMT solvers enables the verification of safety properties for railway network instances of industrial size.

Compared to the previously published work by the same authors [37], the following main extensions have been added: (E-1) the state space encodings and state transition relation are elaborated in more detail, (E-2) new experimental results are presented for the early deployment line (EDL) of the Danish Signalling Programme, and (E-3) benchmarking experiments comparing our BMC approach to other techniques such as BDD-based techniques or CEGAR techniques – by translating our models to nuXmv [9] – are reported.

The remainder of the article is organised as follows: First, in Section 2, some mathematical preliminaries are explained and Section 3 gives a brief introduction to the new Danish route-based interlocking systems. Then the proposed method is presented in Section 4. Next, the five steps constituting the method are described in more detail: (1) – (2) the specification and validation of configuration data are presented in Section 5 and Section 6, respectively; (3) – (4) the generation of Kripke structure models of the behaviours of interlocking systems as well as their desired formal properties, in Section 7 and Section 8, respectively; and (5) the verification strategy in Section 9. Afterwards experimental results with the toolchain as well benchmarking results are shown in Section 10. Finally, related work and concluding remarks are presented in Section 11 and Section 12, respectively.

2. Mathematical Preliminaries

This section explains some mathematical preliminaries that are used in our method, in particular Kripke structures and the k-induction scheme for proving invariants in a Kripke structure.

2.1. Kripke Structures

Kripke structures are used to specify *behavioural models* of interlocking systems in our method. A *Kripke structure* K is a five-tuple (S, s_0, R, L, AP) with state space S, initial state $s_0 \in S$, a total transition relation $R \subseteq S \times S$, and labelling function $L : S \to 2^{AP}$, where AP is a set of atomic propositions and 2^{AP} is the power set of AP. The labelling function L maps a state s to the set $L(s)$ of atomic propositions that hold in s.

Two states s and s' are said to be *consecutive* in K, if there is a transition from s to s', i.e., $(s, s') \in R$. A *path* in K is a finite or infinite sequence of consecutive states. A state s' is said to be *reachable* from another state s in K, if there exists a finite path $s \ldots s'$ starting in s and ending in s'. A state $s \in S$ is said to be *reachable* if it is reachable from the initial state s_0. The *reachable states* of K is the set of all reachable states.

In the context of this article, the states of a Kripke structure are represented by valuation functions $s : V \to D$ over finite sets $V = \{v_0, \ldots, v_n\}$ of variables, where each variable $v_i \in V$ has an associated finite domain D_{v_i}. The range of a state s is $D = \bigcup_{v \in V} D_v$. The whole state space S is the set of all valuation functions $s : V \to D$ for which $s(v) \in D_v$ for all $v \in V$. The *equality* relation $(=)$ between states is defined by the equality of mathematical functions as follows: two states s and s' are equal – denoted by $s = s'$ – iff every variable $v \in V$ is evaluated to the same value in s and s', i.e.,

$$(s = s') \equiv \left(\bigwedge_{v \in V} s(v) = s'(v) \right) \tag{1}$$

For a proposition ϕ over free variables in V, we use $\phi(s)$ to denote the proposition obtained by replacing every occurrence of $v \in V$ in ϕ by the value $s(v)$. A proposition ϕ over free variables in V is said to *hold* in a state $s \in S$, denoted as $s \models \phi$, iff $\phi(s)$ holds. An *invariant* in K is a proposition that holds in all reachable states of K. The initial state s_0 can be represented by the following proposition.

$$\mathcal{I}(s_0) \equiv \bigwedge_{v \in V} v = s_0(v) \tag{2}$$

The transition relation $R \subseteq S \times S$ can also be represented in propositional form as a proposition Φ over free variables in $V \cup V'$ such that

$$R = \{(s, s') \in S \times S \mid \Phi(s, s')\} \tag{3}$$

where $V' = \{v' \mid v \in V\}$ is a duplicate of V used to representing the next state, and $\Phi(s, s')$ is the proposition Φ with every occurrence of $v \in V$ replaced by the value $s(v)$, and every occurrence of $v' \in V'$ replaced by the value $s'(v)$.

A finite path $s_n.s_{n+1} \dots s_{n+k-1}$ of length k through the model K, starting in an *arbitrary* state s_n (which may or may not be reachable), is identified by a solution to the satisfiability of the following proposition.

$$\pi(s_n, \dots, s_{n+k-1}) \equiv \bigwedge_{i=1}^{k-1} \Phi(s_{n+i-1}, s_{n+i}) \tag{4}$$

An *acyclic* path of length k is a finite path $s_n.s_{n+1} \dots s_{n+k-1}$ in which there does not exist a pair of states in the path that are equal. Such a path is characterised by a solution to the satisfiability of the following formula.

$$\pi^{=}(s_n, \dots, s_{n+k-1}) \equiv \pi(s_n, \dots, s_{n+k-1}) \wedge \bigwedge_{n \leq i < j \leq n+k-1} (s_i \neq s_j) \tag{5}$$

2.2. Bounded Model Checking

Model checking can be used to verify that a model of a given system obeys the specification of its intended behaviours. This is done by exploring the state space in an exhaustive, efficient, and highly automatic manner. However, the state space grows exponentially in the number of system components, which is referred to as the *state space explosion problem*. This is the main obstacle preventing model checking from being applied to complex systems in industrial applications. *Bounded model checking* (BMC) remedies this problem by investigating model properties within the vicinity of a state, exploring only those states that are reachable by means of a bounded number of transition steps. Reachability is decided using satisfiability solving, hence avoiding to explore the whole state space [3]. Theoretically, BMC can verify global properties – properties that hold in all reachable states of a system, e.g., invariants as described in the previous subsection – if the transition relation is unrolled for a sufficient number of steps, known as the *recurrence diameter* of the given transition system [3, 5]. In practice, the recurrence diameter of a given transition system is often too large, resulting in exhaustion of memory or unacceptable verification time, because the worst case complexity of the satisfiability solving grows exponentially in the number of unrolling steps.

2.3. k-Induction

In order to avoid unrolling too many transition steps when verifying global properties, *k-induction* – a technique that combines BMC and inductive reasoning – is used.

k-induction Scheme. The following two-step k-induction scheme based on acyclic paths [30, 27] can be applied in order to prove a proposition ϕ is an invariant in K.

(1) *Base Case*: prove that ϕ holds in every state of every acyclic path $s_0.s_1 \dots s_{k-1}$ of length $k > 0$, starting from the initial state s_0, i.e., the following holds, where $\Rightarrow$ is logical implication.

$$(\mathcal{I}(s_0) \wedge \pi^{=}(s_0, \dots, s_{k-1})) \Rightarrow \bigwedge_{i=0}^{k-1} \phi(s_i) \tag{6}$$

(2) *Induction Step*: prove that if ϕ holds in every state of an acyclic path $s_n.s_{n+1} \dots s_{n+k-1}$ of length $k > 0$, starting from an arbitrary state s_n, then ϕ will also hold in every $(k+1)^{th}$ state s_{n+k}. In other words, the following holds

$$\left(\pi^{=}(s_n, \dots, s_{n+k}) \wedge \bigwedge_{i=0}^{k-1} \phi(s_{n+i})\right) \Rightarrow \phi(s_{n+k}) \tag{7}$$

Transformation to Bounded Model Checking Problems. Both the base case and the induction step can be transformed to bounded model checking problems of finding witnesses for the violations. Violations of the base case are identified by the negation of Formula 6 as shown in the following.

$$\mathcal{I}(s_0) \wedge \pi^{=}(s_0, \dots, s_{k-1}) \wedge \neg \bigwedge_{i=0}^{k-1} \phi(s_i) \tag{8}$$

A solution, if found, for Formula 8 identifies an execution of the system in which ϕ does not hold in at least one state within the vicinity of k transition steps from the initial state s_0. Likewise, violations of the induction step are identified by the negation of Formula 7 as shown in the following.

$$\pi^{=}(s_n, \dots, s_{n+k}) \wedge \bigwedge_{i=0}^{k-1} \phi(s_{n+i}) \wedge \neg\phi(s_{n+k}) \tag{9}$$

A solution, if found, for Formula 9 shows an execution of length $(k+1)$ of the system where ϕ holds for the first k states, but not in the last one. If no violation is found for the base case or the induction step, then ϕ is an invariant in K.

Strengthening Invariants. As pointed out in [27], when ϕ is not strong enough to be inductive, counterexamples are found for the induction case. If ϕ is indeed an invariant in K, then these counterexamples are *spurious*, i.e., they start from an unreachable state and do not correspond to any actual run of the considered system. In order to make ϕ inductive, it is strengthened with an extra invariant ψ, i.e., in the k-induction proof, one should prove $\phi \wedge \psi$ instead of ϕ. ψ is called the *strengthening invariant*, which eliminates the spurious counterexamples. An example of a strengthening invariant is given in Section 9.

An alternative to invariant strengthening is to increase k until reaching a value where one of the following occurs: (1) the base case fails, i.e., the property does not hold; or (2) both the base case and the induction step succeed, i.e., the property holds. According to [27], any k-induction proof can be reduced to a 1-induction proof by means of invariant strengthening. In this article, we decided not to use the alternative method, as our experiments in Section 10 showed that increasing k had severe penalty on the verification performance due to the state space explosion problem.

3. The new Danish Route-based Interlocking Systems

In this section we introduce briefly the new Danish interlocking systems and the domain terminology. The subsequent subsections explain (1) different components of a specification of an interlocking system which is compatible with ETCS Level 2, (2) the interlocking principles and the strict procedure that interlocking systems employ to ensure safety, and (3) the sequential release feature, respectively.

3.1. Specification of Interlocking Systems

The specification of a given route-based interlocking system consists of two main components: (1) a railway network, and (2) a corresponding interlocking table.

Railway Networks. A railway network in ETCS Level 2 consists of a number of track-side elements of different types:[5] linear sections, points, and marker boards. Figure 1 shows an example layout of a railway network having six linear sections (`b10`,`t10`,`t12`,`t14`,`t20`,`b14`), two points (`t11`,`t13`), and eight marker boards (`mb10`, ..., `mb21`). These terms, as well as their functionality within the railway network, are explained in more detail in the next paragraphs.

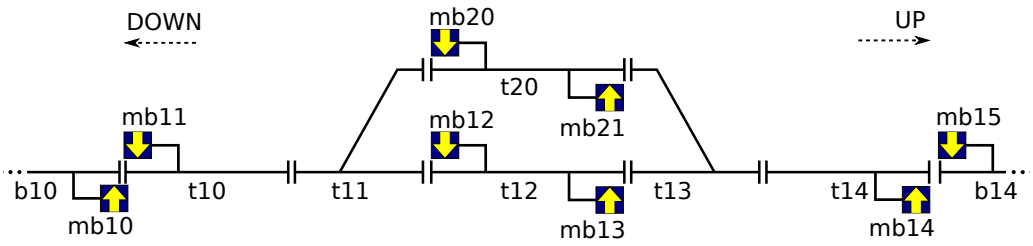


Figure 1: An example railway network layout.

A *linear section* is a section with up to two neighbours: one in the *up* end, and one in the *down* end. For example, the linear section `t12` in Figure 1 has `t13` and `t11` as neighbours at its up end and down end, respectively. In Danish railway's terminology, *up* and *down* denote the directions in which the distance from a reference location is *increasing* and *decreasing*, respectively. The reference location is the same for both up and down, e.g., an end of a line. For simplicity, in the examples and figures in the rest of this article, the *up* (*down*) direction is assumed to be the left-to-right (right-to-left) direction, if it is not indicated otherwise.

A *point* can have up to three neighbours: one at the *stem*, one at the *plus* end, and one at the *minus* end, e.g., point `t11` in Figure 1 has `t10`, `t12`, and `t20` as neighbours at its stem, plus, and minus ends, respectively. The ends of a point are named so that the *stem* and *plus* ends form the straight (main) path, and the *stem* and *minus* ends form the branching (siding) path. A point can be switched between two positions: PLUS and MINUS. When a point is in the PLUS (MINUS) position, its *stem* end is connected to its *plus* (*minus*) end, thus traffic can run from its *stem* end to its *plus* (*minus*) end and vice versa. It is not possible for traffic to run from *plus* end to *minus* end and vice versa.

Linear sections and points are collectively called (train detection) sections, as they are provided with train detection equipments used by the interlocking system to detect the presence of trains on the sections. Note that sections are bidirectional by default, i.e., trains are allowed to travel in both directions (but not at the same time) in a given section. For instance, trains can travel a linear section from its up end to its down end, and from its down end to its up end.

Along each linear section, up to two *marker boards* (one for each direction) can be installed. A marker board can only be seen in one direction and is used as reference location (for the start and end of routes) for trains going in that direction. For example, in Figure 1, marker board `mb13` is installed along section `t12` for travel direction up. Contrary to legacy systems, there are no physical signals in ETCS Level 2, but interlocking systems have a *virtual signal* associated with each marker board. Virtual signals play a similar role as physical signals in legacy systems: a virtual signal can be OPEN or CLOSED, respectively, allowing or disallowing traffic to pass the associated marker board. However, trains (more precisely train drivers) do not see the virtual signals, as opposed to physical signals. Instead, the aspect of virtual signals (OPEN or CLOSED) are communicated to the onboard computer in the train via a radio network. For simplicity, the terms *virtual signals*, *signals*, and *marker boards* are used interchangeably throughout this article.

[5]Here we only show types that are relevant for the work presented in this article.

Interlocking Tables. An interlocking system monitors constantly the status of track-side elements, and sets them to appropriate states in order to allow trains traveling safely through the railway network under control. The new Danish interlocking systems are route-based. A *route* is a path from a *source* signal to a *destination* signal in the given railway network. A route is called an *elementary route* if there are no signals that are located between its source signal and its destination signal, and that are intended for the same direction as the route.

In railway signalling terminology, *setting* a route denotes the process of allocating the resources – i.e., sections, points, and signals – for the route, and then *locking* it exclusively for only one train when the resources are allocated. On the other hand, *releasing* a route denotes the process of releasing the resources that have been allocated for a route after they have been used by a train.

An *interlocking table* specifies the elementary routes in the given railway network and the conditions for setting these routes. The specification of a route r and conditions for setting r include the following information:

- $id(r)$ – the route's unique identifier,
- $src(r)$ – the source signal of r,
- $dst(r)$ – the destination signal of r,
- $path(r)$ – the list of sections constituting r's path from $src(r)$ to $dst(r)$,
- $overlap(r)$ – a list of the sections in r's overlap, i.e., the buffer space after $dst(r)$ that would be used in case trains overshoot the route's path,
- $points(r)$ – a map from points[6] used by r to their required positions,
- $signals(r)$ – a set of protecting signals used for flank or front protection [32] for the route, and
- $conflicts(r)$ – a set of conflicting routes which must not be set while r is set.

Table 1 shows an excerpt of an interlocking table for the network shown in Figure 1. Each row of the table corresponds to a route specification. The column names indicate the information of the route specifications that these columns contain. As can be seen, one of the routes has id `1a`, goes from `mb10` to `mb13` via three sections `t10`, `t11` and `t12` on its path, and has no overlap. It requires point `t11` (on its path) to be in PLUS position, and point `t13` (outside its path) to be in MINUS position (as a protecting point). The route has `mb11`, `mb12` and `mb20` as protecting signals, and it is in conflict with routes `1b`, `2a`, `2b`, `3`, `4`, `5a`, `5b`, `6b`, and `7`.

Table 1: Excerpt of the interlocking table for the network layout in Figure 1. The overlap column is omitted as it is empty for all of the routes. (`p` means PLUS, `m` means MINUS.)

id	src	dst	path	points	signals	conflicts
1a	mb10	mb13	t10;t11;t12	t11:p;t13:m	mb11;mb12;mb20	1b;2a;2b;3;4;5a;5b;6b;7
..	...	...	...	...	...	...
7	mb20	mb11	t11;t10	t11:m	mb10;mb12	1a;1b;2a;2b;3;5b;6a

[6] These include points in the path and overlap, and points used for flank and front protection. For detail about flank and front protection, see [32].

3.2. Interlocking Principles

In order to prevent hazardous situations, e.g., collision and derailment of trains, route-based interlocking systems employ a classic principle: *a route is locked exclusively for use of one train at a time*. This is obtained by following a strict procedure for setting and releasing routes based on information in their interlocking tables. As an example, let us consider the following procedure for route `1a` specified in Table 1:

(0) Initially the route is *free*.

(1) The route is dispatched either manually by a signalman or automatically by a traffic management system. As a result, the route is *marked* as requested.

(2) The interlocking system checks the status of different track-side elements in the system to figure out whether it can start *allocating* resources for route `1a`, e.g., sections `t10`, `t11` and `t12` must be vacant, and the conflicting routes must not be allocated or locked. If so, the interlocking commands the protecting signals of the route – i.e., `mb11`, `mb12` and `mb20` – to change to CLOSED, and it commands points to switch to their required positions according to the route's specification – i.e., it commands `t11` to switch to PLUS, and `t13` to switch to MINUS.

(3) The interlocking system monitors constantly the status of the track-side elements. When the signals and points have changed their status as commanded in step (2), the route is *locked* and its source signal `mb10` is commanded to change to OPEN, allowing a train to enter the route.

(4) When the locked route is *occupied* – i.e., a train enters it, the source signal `mb10` is set to CLOSED preventing other trains from entering.

(5) The whole route is *released* (set back to *free*) when the train has finished using it – i.e., the train has passed `mb13`, or the train has come to a standstill in `t12` in front of `mb13`.

3.3. Sequential Release

The new Danish interlocking systems employ *sequential release* (also known as sectional release) [32, chap. 4]. This feature results in two major changes to the procedure explained above:

(a) With sequential release, the interlocking can release an element in a locked route as soon as the train has passed it, instead of waiting until the train has finished using the route and then releasing the route as a whole. Consequently, the capacity increases.

(b) As a direct result of (a), a route may be allocated – in step (2) of the procedure in Section 3.2 – while some of its conflicting routes are still occupied by trains, instead of waiting for all of its conflicting routes to be released as specified in the procedure.

The advantage of sequential release is better illustrated by an extended example as shown in Figure 2 where one or several linear sections have been added between sections `t11` and `t12` on route `1a` in Table 1. For this example, as soon as a train `T1` has left `t11` while going along route `1a`, `t11` can be released. As a consequence, route `7` (see Table 1) can already be *allocated* by another train `T2` (assuming that other allocation conditions are fulfilled), while `1a` is still occupied by `T1`. If sequential release had not been allowed, route `7` could first have been *allocated*, when route `1a` had been released (i.e., when `T1` had passed `mb13`, or it had come to a standstill in `t12` in front of `mb13`).

Note that the term *sequential release* may be used differently at different operational levels. (1) At a high operational level, sequential release is performed on compound routes (i.e., routes created by chaining multiple elementary routes together so that all of them can be set in one go): each single elementary route of a given compound route is released sequentially. In that sense, sequential release is a *route-wise* feature. (2) On the other hand, at a low operational level, sequential release is performed on elementary routes: each single element of an elementary route is released sequentially [32, chap. 4]. In this case, sequential release is an *element-wise* feature. In the work documented in this article, we consider sequential release in its latter usage of the term. In order to include this feature into the models of interlocking systems, additional variables and transitions are required as described in Section 7 below. Therefore, the models become more complex with a higher level of concurrency. Consequently, the verification tasks are more challenging.

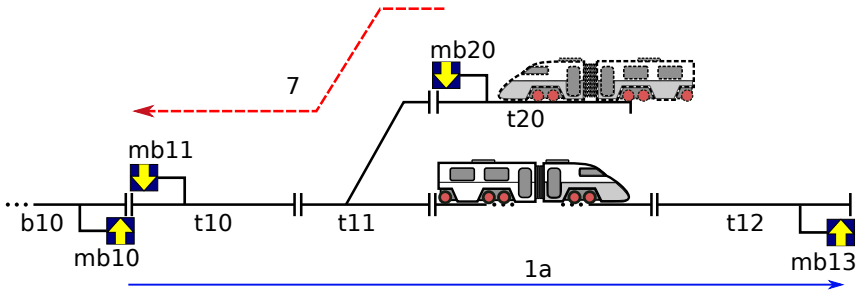


Figure 2: Extended example illustrating sequential release's advantage.

4. Method Overview

Our method for modelling and verification of railway interlocking systems is a combination of domain-specific approaches and formal methods. According to this, an environment consisting of the following components is provided.

(a) A generic (configurable) behavioural model of the Danish interlocking systems and associated generic (configurable) desired properties. These properties include generic safety properties, and generic strengthening invariants used for the k-induction scheme described in Section 2.

(b) A *domain-specific language* (DSL) that is specially tailored for the railway interlocking domain [36]. In this language one can specify configuration data consisting of a network layout and an interlocking table for an interlocking system, as described in Section 3.

(c) A DSL *specification editor* and *static checker* for configuration data of interlocking systems.

(d) A *model generator* that takes the generic model and a well-formed DSL specification of configuration data as input, and produces a concrete behavioural model of the given interlocking system as output.

(e) A *properties generator* that takes the description of generic properties and a well-formed DSL specification of configuration data as input, and produces concrete properties as output.

(f) A *bounded model checker* that can perform k-induction.

The modelling and verification process uses this environment and consists of the following steps as illustrated in Figure 3. The steps (1)–(5) are further elaborated in subsequent sections, Section 5–Section 9, respectively.

(1) A DSL specification of the configuration data (a network layout and its corresponding interlocking table) is constructed.

(2) The static checker verifies whether the configuration data is statically well-formed according to the static semantics of the DSL.

(3) The model generator instantiates the generic model with well-formed configuration data. This results in a model instance in the form of a Kripke structure.

(4) Similarly, the properties generator instantiates the generic properties. This results in concrete properties expressed as state invariants in the Kripke structure generated in step (3).

(5) The generated model instance is then checked against the concrete properties by the bounded model checker using a combination of BMC and inductive reasoning as described in Section 2. If the model instance does not satisfy the properties, counterexamples will be generated. An interface for visualising the counterexamples at the DSL level is integrated into the editor in Eclipse, see [18].

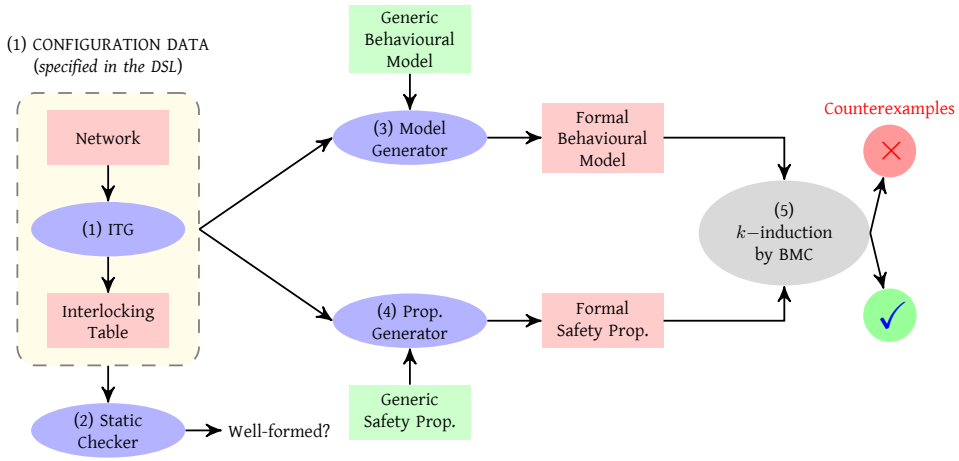


Figure 3: Verification method and the associated toolchain.

The toolchain associated with the method has been implemented using the RT-Tester framework [28, 34]. The bounded model checker in RT-Tester uses the SONOLAR SMT solver [29] to compute counterexamples showing the violations of the base case or induction step. RT-Tester has been selected because (1) it is an integrated model-based testing and BMC tool, and (2) its SMT solver also supports floating point arithmetic. The first property is crucial for us, because our objective is to complement the model verification with HW/SW integration tests. The second capability is vital, because we also plan to extend the model by real-time aspects, such as train velocity and braking curves.

5. Interlocking Configuration Data Construction

This section describes how interlocking configuration data is constructed in step (1) of the modelling and verification process described in Section 4.

We have developed a *domain-specific language* (DSL) for specifying configuration data for route-based interlocking systems that are compatible with ETCS Level 2. The configuration data that can be described in this language consists of the specification components that were described in Section 3, i.e., a DSL specification consists of a network layout and an associated interlocking table. The DSL has been formally specified by the authors in a previous publication, see [36], and an eXtensible Markup Language (XML) representation of the language has been implemented as a front-end of our toolchain. The XML representation of some sample networks can be found at `http://www.imm.dtu.dk/~aeha/RobustRailS/data/casestudy`, and at the end of this section, an excerpt of the XML representation of the railway network layout shown in Figure 1 and the XML representation of the interlocking table shown in Table 1 can be found.

A graphical user interface for the language has also been implemented [18].

In the first step of our modelling and verification process, a DSL specification of configuration data in XML format should be created. The specification can be created manually, or exported from computer-aided design tools supporting the XML format. Alternatively, the user can use the graphical user interface [18] to specify the configuration data by drawing the network layout and type in the interlocking table via an editor implemented as an Eclipse plug-in. The editor can then export the specification to the XML format.

The interlocking table generator (ITG) specified in [36] has also been implemented in our toolchain. As an option the user may not explicitly provide an interlocking table, but only a network layout and then use the ITG to get a complete interlocking table (for all possible routes) including route protection [32] (by protecting signals and/or protecting points) created automatically from the network layout [36].

Example of XML representation. The XML representation of a network layout consists of a description of each track section and a description of each marker board in the network. Here is an excerpt of the XML representation of the railway network layout shown in Figure 1:

```
<network id="mininetwork">
  <trackSection id="b10" length="100" type="linear">
    <neighbor ref="t10" side="up"/>
  </trackSection>
  <trackSection id="t10" length="87" type="linear">
    <neighbor ref="b10" side="down"/>
    <neighbor ref="t11" side="up"/>
  </trackSection>
  <trackSection id="t11" length="26" type="point">
    <neighbor ref="t10" side="stem"/>
    <neighbor ref="t12" side="plus"/>
    <neighbor ref="t20" side="minus"/>
  </trackSection>
  ...

  <markerboard id="mb10" track="b10" mounted="up" distance="50"/>
  <markerboard id="mb11" track="t10" mounted="down" distance="50"/>
  ...
</network>
```

As it can be seen, for each track section there is a description of its *id*, its *length*, its *type* (whether it is a linear section or a point), and its *neighbor*-sections on each side, as far as defined. For each marker board there is a description of its *id*, the *track* section at which it is placed, the travel *direction* in which it is *mounted*, and the *distance* from the marker board to the end of the track section.

The XML representation of an interlocking table consists of a specification of each row in the table. Here is an XML representation of the first row of the interlocking table shown in Table 1:

```
<route id="1a" source="mb10" destination="mb13">
  <condition ref="t10"  type="trackvacancy"/>
  <condition ref="t11"  type="trackvacancy"/>
  <condition ref="t12"  type="trackvacancy"/>
  <condition ref="t13"  type="point" val="minus"/>
  <condition ref="t11"  type="point" val="plus"/>
  <condition ref="mb11" type="signal"/>
  <condition ref="mb12" type="signal"/>
  <condition ref="mb20" type="signal"/>
  <condition ref="1b" type="mutualblocking"/>
  <condition ref="2a" type="mutualblocking"/>
  <condition ref="2b" type="mutualblocking"/>
  <condition ref="3"  type="mutualblocking"/>
  <condition ref="4"  type="mutualblocking"/>
  <condition ref="5a" type="mutualblocking"/>
  <condition ref="5b" type="mutualblocking"/>
  <condition ref="6b" type="mutualblocking"/>
  <condition ref="7"  type="mutualblocking"/>
</route>
```

In the first line, it can be seen how the *id*, the *source* marker board and the *destination* marker board of the route are specified. In the following lines, the remaining information about the route is represented in the

form of conditions of different types: Conditions of type `"trackvacancy"` contain references to the track sections constituting the *path* of the route. Conditions of type `"point"` contain references to the *points* together with their required position. Conditions of type `"signal"` contain references to the protecting *signals*. Conditions of type `"mutualblocking"` contain references to the conflicting routes.

6. Interlocking Configuration Data Validation

A static checker for validating interlocking configuration data in step (2) of the modelling and verification process described in Section 4 has been implemented in our toolchain. The static checker checks whether a given DSL specification of configuration data (constructed in step (1)) is well-formed with respect to the static semantics [36] of the DSL. For example, the static checker checks whether conflicting routes are correctly listed in the interlocking table, and whether every route in the specification has proper protection provided by protecting signals and/or protecting points. If the checker discovers errors in the configuration data, suggestions for fixing them will be provided to the user. Examples of such suggestions can be to add protecting signals/points, or to list physically conflicting routes when such information is missing from the given interlocking table.

For instance, if route `7` were not listed in the conflicts of route `1a` in Table 1, the following error message would be given:

```
Routes 1a and 7 are in conflict, but route 7 is not listed in the conflicts of route 1a.
Reasons to be in conflict: Non-concatenated routes with shared elements: t10, t11.
```

And if signal `mb11` were not listed in the protecting signals of route `1a`, the following error message would be given:

```
For route 1a, signal mb11 at section t10 should have been listed as a protecting signal.
```

7. Behavioural Models Generation

This section describes how formal, behavioural models of the new Danish interlocking systems are generated in step (3) of the modelling and verification process described in Section 4.

We have developed a *generic model* [35] (in the form of a model pattern) of the Danish interlocking systems and a *model generator* that can be used to instantiate this generic model with configuration data specified in the DSL described above.

In step (3) of our method, this generator is applied to the configuration data that was created in step (1) and verified to be statically well-formed in step (2). This results in a model instance in the form of a Kripke structure. Due to the complexity of the Kripke structure encodings, in the following subsections, we only outline how the state space S, the initial state s_0, and the transition relation in propositional form Φ of the resulting Kripke structure are obtained from the configuration data consisting of a network layout N and an interlocking table T. The model instances that are obtained by applying the generator to the sample networks mentioned in Section 5 can be found at `http://www.imm.dtu.dk/~aeha/RobustRailS/data/casestudy`.

Through the rest of this article, for readability, named constants and their corresponding integral values are used interchangeably. We use the notation *name(integral-value)* to mean that *name* is the name of a constant having the value *integral-value*. For instance, PLUS(0) denotes a constant PLUS having the value of 0. Furthermore, integral values prefixed with $0b$ are written in their binary forms, e.g., $0b110$ is the binary value 110 (which is 6 in decimal).

7.1. State Space

As described in Section 2, the state space S is the set of all valuation functions $s : V \to \bigcup_{v \in V} D_v$ for which $s(v) \in D_v$ for all $v \in V$, where V is a set of variables used to represent the status of different components, such as track elements and routes, in the given interlocking system. Each variable $v \in V$ has an associated finite domain $D_v \subset \mathbb{N}_0$. The subsequent paragraphs present the set of variables V and their associated value domains D_v.

Occupancy Status. As mentioned in Section 3, trains can travel in any physically possible direction on a section. Thus, for each section in the network layout N, we use, for each physically possible travel direction, a variable to keep track of the occupancy status of that section in that possible travel direction. For instance, for each linear section l in N, the variable $l.U2D$ records the occupancy status of l in the direction from its up end to its down end, while the variable $l.D2U$ records the occupancy status in the opposite direction. Similarly, for each point p in N, three variables $p.S2PM$, $p.P2S$, and $p.M2S$ record the occupancy status of the point in the direction from its stem end to its plus/minus ends, from its plus end to its stem end, and from its minus end to its stem end, respectively. The movement of trains through a given network are reflected by state transitions of the occupancy status variables of the sections in the network. This is specified in detail in Section 7.7 below.

The occupancy status of a section in a given travel direction is encoded using the three least significant bits `HTO` of a non-negative integer variable as shown in Figure 4. The value `1` of the bits `H`, `T`, `O` indicate: (`H`) the head of the train is within the section, (`T`) the tail of the train is within the section, and (`O`) the section is occupied, respectively. Note that in some cases the encoding contains redundant information, e.g., a section is obviously occupied when the head of a train is inside the section. However, the `O` bit is necessary for modelling the case where a train is occupying the section, but the head and the tail are outside the section, as it is the case for section *t3* in Figure 5.

This occupancy status encoding offers two advantages: (a) the encoding can cover the case where a train occupies more than one section (e.g., when it is crossing the joint between two sections), and (b) the safety properties can be expressed efficiently using arithmetic operations on integer variables as will be shown in Section 8.

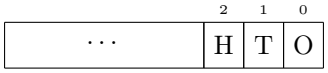


Figure 4: A variable recording occupancy status of a section.

Figure 5 shows some examples of how different values of occupancy status variables reflect how a train occupies sections. Let us assume that the train is travelling in the direction down-to-up. Section *t1* is occupied by the whole train, thus its occupancy status variable $t1.D2U$ is $0b111$ indicating that all three bits H, T, and O are set. *t2* is occupied by the tail of a train, thus its occupancy status variable is $t2.D2U = 0b011$, *t3* is occupied without a head or a tail of a train in it – i.e., the train is long enough to cover the whole section, thus $t3.D2U = 0b001$. Lastly, *t4* is occupied by a head of a train, therefore its occupancy status variable is $t4.D2U = 0b101$.

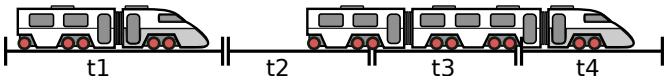


Figure 5: Example of sections having different occupancy status.

A section is vacant when all of its occupancy status variables are evaluated to zero. For example, a linear section l is vacant when both $l.D2U$ and $l.U2D$ are evaluated to zero. Since all occupancy status variables are non-negative, we can define the following formula to determine whether a linear section l is vacant.

$$vacant(l) \equiv (l.D2U + l.U2D = 0) \tag{10}$$

An analogous formula can be defined for a point.

Lockable Elements. In order to accommodate sequential release in our model, we consider a linear or point section as a *lockable element*. For each lockable element e in the network layout N, there are two variables for representing the status of e:

- $e.MODE$ – recording the mode of the element, and

- $e.PREV$ – recording whether the previous section in the route has been released.

The possible values of $e.MODE$ are: FREE (the element is not *exclusively* locked by a route, or used by any train), EXLCK (the element is *exclusively* locked for a route), or USED (the element has been used, i.e., occupied, by a train after it was *exclusively* locked for a route). The possible values of $e.PREV$ are: PENDING(0) and RELEASED(1).

Point Positions. For each point p in the network layout N, there are two variables:

- $p.POS$ – for representing the actual position of the point, and
- $p.CMD$ – for representing the point position commanded by the interlocking.

The possible values of $p.POS$ are: PLUS(0), MINUS(1), and INTERMEDIATE(2) – the position where the point is switching from plus (minus) to minus (plus). The possible values of $p.CMD$ are only PLUS and MINUS, as the interlocking *cannot* command a point to switch to the INTERMEDIATE position.

Signal Aspects. For each virtual signal s in the network layout N, there are two variables:

- $s.ACT$ – for representing the actual aspect of the signal as "seen" by the train, and
- $s.CMD$ – for representing the aspect of the signal as commanded by the interlocking system.

The possible values of these variables are: OPEN and CLOSED. The values of these two variables may differ due to the delay in the communication between the interlocking system and the onboard computers in the trains.

Routes. For each route r in the interlocking table T, there is a variable $r.MODE$ to represent the current mode of that route. A route can be in one of the following modes: FREE, MARKED, ALLOCATING, LOCKED, or OCCUPIED.

7.2. Initial State

The initial state s_0 is the state in which all sections are FREE and vacant (i.e., there are no trains in the network), the commanded and actual aspects of all signals are CLOSED, all routes are FREE, and the commanded and actual positions of all points are PLUS. Our encodings in Section 7.1 are deliberately chosen so that s_0 is the state in which all variables are evaluated to 0. Therefore, the propositional form of the initial state is

$$\mathcal{I}(s_0) \equiv \bigwedge_{v \in V} v = 0 \tag{11}$$

7.3. Transition Relation

This section describes how the transition relation in propositional form Φ is constructed. The general construction principles are as follows:

(a) Propositions (also called transition rules) specifying transitions for atomic events, referred as *atomic transitions*, are defined. They take the form $guard \wedge effect$, where *guard* is a proposition over the variables in V describing the precondition for the transition and *effect* is a proposition over free variables from $V \cup V'$ specifying how the values of the variables are changed by the transition. The *guard* and the *effect* propositions are obtained by instantiating generic proposition patterns with data from the given network layout and the associated interlocking table.

Exactly one atomic transition is taken at a time, i.e., interleaving semantics is used. When an atomic transition is taken, all changes in the values of variables in *effect* occur in one time-step, as a shared memory, cycle-based software execution model is used. This model fits well with the realistic execution of interlocking controllers for two reasons. First, an interlocking controller and its environment are

concurrent processes communicating over shared variable interfaces. Second, in practice, the execution of interlocking controllers is divided into cycles. In each cycle, the controllers read their inputs, make decision based on their control algorithm, and then write changes to the outputs. Inputs do not change within a cycle, and outputs are not accessible until the cycle ends.

(b) Different types of events/transitions are assigned priorities because different types of events in interlocking systems occur at significantly different speed and we prefer to avoid introducing time into the model.

(c) Propositions for atomic events are combined according to their relative priorities. If two transitions having different priorities are enabled at the same time, the one with higher priority should be taken. On the other hand, transitions with the same priority should be chosen nondeterministically, if they are enabled at the same time. Therefore, if transitions represented by propositions Φ_a and Φ_b have the same priority, their combined behaviour is described by the proposition $\Phi_a \vee \Phi_b$. On the other hand, if the transitions represented by Φ_a has higher priority than the ones represented by Φ_b, then their combined behaviour – denoted as $\Phi_a\ [>]\ \Phi_b$ – is described by $\Phi_a \vee (\neg g_a \wedge \Phi_b)$, where g_a is the condition for at least one of the transitions in Φ_a to be enabled. Thus

$$\Phi_a\ [>]\ \Phi_b \equiv \Phi_a \vee (\neg g_a \wedge \Phi_b) \tag{12}$$

Note that $[>]$ operator is right-associative, i.e.,

$$\Phi_a\ [>]\ \Phi_b\ [>]\ \Phi_c \equiv (\Phi_a\ [>]\ (\Phi_b\ [>]\ \Phi_c)) \tag{13}$$

Overview of Transitions and their Priorities. We group the transitions of atomic events of an interlocking system into four types, such that the transitions of each type have the same priority and their propositions can be combined using the $\vee$ operator, resulting in a proposition for the combined behaviour of the transitions of that type. The types and their combined propositions are as follows:

(0) Φ_δ – representing route dispatching transitions;

(1) Φ_ι – representing interlocking controller transitions, e.g., setting the mode of a route;

(2) Φ_ϵ – representing track element transitions, e.g., switching a point, or communicating a signal aspect to a train; and

(3) Φ_τ – representing train movement transitions.

Transitions of type (0) are not prioritised, i.e., they can be taken nondeterministically whenever they are enabled, independently from other transitions. On the other hand, transitions of types (1), (2), and (3) are prioritised in the descending order that they appear in the list, i.e., transitions of type (1) have the highest priority and transitions of type (3) have the lowest. This priority of transitions is based on the intuition that in practice, the events in the interlocking controller occur at significantly higher speed than the ones occurring in a track element. For example, a cycle of a route controller occurs within a hundred milliseconds, while switching a point from one position to another may take a few seconds. An analogous argument applies to events related to track elements compared to events related to train movements. For instance, switching a point may take a few seconds, while it may take a train minutes to pass a section.

With this grouping, Φ_ι models the behaviours of the system under consideration – the interlocking controller; while Φ_δ, Φ_ϵ, and Φ_τ model the behaviours of the operational environment that interacts with the interlocking controller. The interlocking system is a closed system that evolves according to the combined transition relation Φ as specified in the following.

$$\Phi \equiv \Phi_\delta \vee (\Phi_\iota\ [>]\ \Phi_\epsilon\ [>]\ \Phi_\tau) \tag{14}$$

The route dispatching transitions Φ_δ is given the same priority as the priority given to the combined transition relation specifying all the other transitions in order to allow routes to be dispatched arbitrarily. If route dispatching transitions were given higher priority than the one given to any of the other transitions, all routes which could be dispatched would have to be dispatched before interlocking controller, track elements, or trains could make any transitions. On the other hand, if route dispatching were given lower priority than any of the other transitions, then a route could not be dispatched, if another route is processed by the interlocking controller, or a track element or a train could make a transition.

In the subsequent subsections, we highlight the essence of the transitions of each type.

7.4. Route Dispatching Transitions

The route dispatching transitions of a given interlocking system, represented by Φ_δ, model how routes are dispatched as the result of manual requests from signalmen or automatic requests from a traffic management system. Figure 6 shows the life-cycle of a route, i.e., its different modes and the transitions from one mode to another. The life-cycle of a route is described in detail in the next subsection. The route dispatching is modelled by the transition labelled (1) in Figure 6. A route can be dispatched arbitrarily whenever its mode is FREE. This means that multiple routes can be dispatched before any of them are processed by the interlocking controller, and routes can be dispatched when other routes are being processed. Upon dispatching, the route's mode changes from FREE to MARKED.

Hence, for each route r in the interlocking table T, the proposition $dispatch(r)$ specifying the dispatching of that route is specified as follows.

$$r.\text{MODE} = \text{FREE} \wedge r.\text{MODE}' = \text{MARKED} \tag{15}$$

Since the objective of the verification is to check safety properties, and not to analyse the effect of specific train sequences following a given time table, we can over-approximate the train behaviour. Therefore, in any system state, it is assumed that every route may be requested by a train in a nondeterministic way, so the transition from FREE to MARKED is always enabled. This leads to a super-set of the model state sequences that could really occur in the real system. Since we are able to prove safety for this super set, the practically infeasible sequences do not present a problem. On the contrary, by making the model more nondeterministic than any real behaviour that might occur in practice, the proposition representing the transition relation is effectively simplified, because state-dependent decisions are over-approximated by nondeterministic ones.

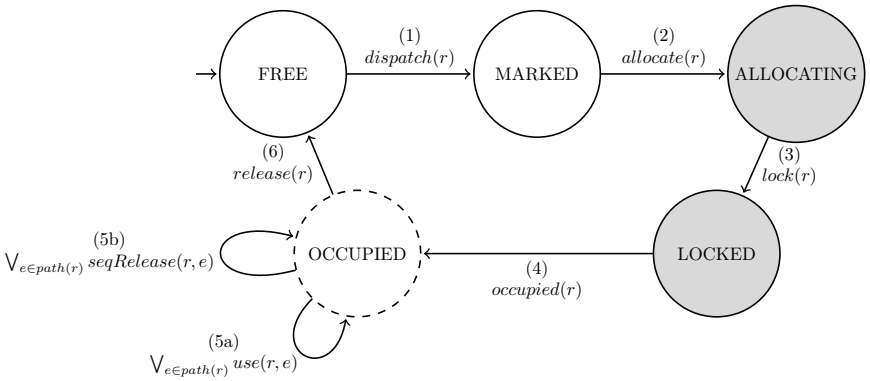


Figure 6: Life-cycle for route r, showing how the value of $r.MODE$ is changed.

7.5. Interlocking Controller Transitions

The interlocking controller transitions of a given interlocking system, represented by Φ_ι, model the internal behaviours of the interlocking controller. Specifically, they model how the status of routes and lockable elements change as response to the changes in the environment. The subsequent paragraphs outline

the life cycle of a route, and a life cycle of a lockable element internally to the interlocking controller, respectively.

Life-cycle of a Route. The life-cycle shown in Figure 6 reflects the procedure for setting and sequentially releasing a route r as described in Section 3.2. The transitions labelled (1), (2), (3), (4), and (6) in Figure 6 correspond to items (1) – (5) in the procedure presented in Section 3.2 for setting and releasing a route. Transitions (5a) and (5b) model the sequential release that can take place while the route stays in OCCUPIED mode: as the train moves along the route, its elements are used (5a) when the train enters them, and then they are released sequentially (5b) as soon as the train has passed them. Transition (2) is adapted to sequential release: allocating resources for a given route r is now also allowed even if a conflicting route r' is in the OCCUPIED mode, given that the elements shared between r and r' have been sequentially released.

For each route r in the interlocking table T, for each of the transitions (2), (3), (4), (5a), (5b), and (6), there is a proposition specifying the transition in a similar way as transition (1) was specified in Formula 15.

Life-cycle of a Lockable Element. Figure 7 depicts the life-cycle of a lockable element e within the network controlled by a given interlocking system. Each node in the diagram in Figure 7 is labelled with the following information about the status of the element e, and the status information is shown in bold when it differs from the corresponding status of the previous node.

(a) $vacant(e)$ – indicating whether the element is vacant where $vacant(e)$ is a formula over occupancy status variables of e as shown in Section 7.1;

(b) its current mode – i.e., the value of $e.MODE$ as described in Section 7.1; and

(c) the value of the $e.PREV$ variable indicating whether the previous element $prev(r, e)$ of e in the route r has been released

An element e is initially in a state where it is vacant, in FREE mode, and its $PREV$ variable is PENDING. (1) When the interlocking controller is allocating a route r that uses e, the mode of the element is set to EXLCK, meaning that the element is locked exclusively for r. (2) The element becomes occupied, i.e., not vacant, as a train enters. (3) The interlocking controller detects the change in the occupancy status of e, consequently sets e's mode to USED. (4) When the train leaves the previous element $prev(r, e)$ of e in the route r, $prev(r, e)$ is released, and it informs e by setting the variable $e.PREV$ to RELEASED. (5) When the train leaves e, the element becomes vacant again. (6) The interlocking controller again detects the change, releases e and informs the next element $next(r, e)$ in the same route by setting $next(r, e).PREV$ to RELEASED.

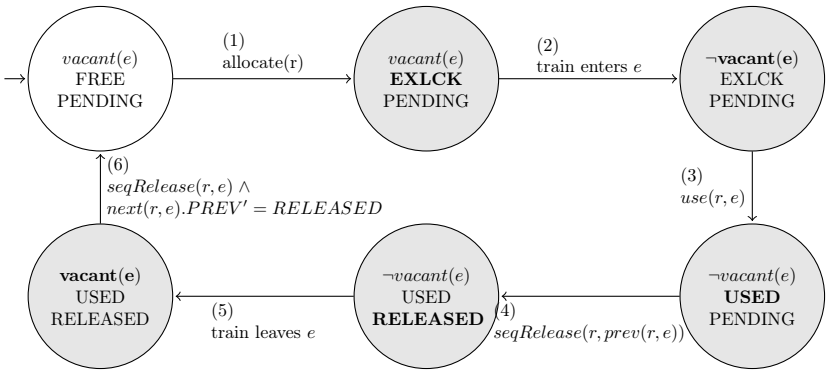


Figure 7: A life-cycle of a lockable element e.

7.6. Track Element Transitions

The track elements transitions of a given interlocking system, represented by Φ_ϵ, model the behaviour of track-side elements in the railway network under control, i.e., how points are switched and how signal aspects are communicated from the interlocking controller to the onboard computer in a train.

Switching Points. A point p can be switched if it is commanded to be switched to a position $p.CMD$ that is different from its current position $p.POS$. The point switching process occurs in two steps: First the point moves from its current position to the *intermediate* position, and then the point is switched from the *intermediate* position to the requested position. Hence, for each point p in the network layout N, there are two transition rules for switching the point:

$$p.POS \neq p.CMD \wedge p.POS \neq INTERMEDIATE \wedge p.POS' = INTERMEDIATE \tag{16}$$

$$p.POS = INTERMEDIATE \wedge p.POS' = p.CMD \tag{17}$$

Communicating Signal Aspects. For each signal s in the network layout N, there is a transition rule for changing the setting of the signal:

$$s.ACT \neq s.CMD \wedge s.ACT' = s.CMD \tag{18}$$

It states that whenever the actual aspect $s.ACT$ of the signal s differs from its commanded aspect $s.CMD$, the actual aspect of the signal is updated to the commanded aspect.

7.7. Train Movements Transitions

Trains are not explicitly specified in our model, in the sense that there are no explicit train objects. Instead, train movements and related aspects are implicitly modelled via the occupancy status of sections, inspired by the "rubber-band" model described in [1]. This implicit model is advantageous compared to the explicit one, because it can model arbitrary numbers of trains of arbitrary length. Therefore, we do not have to investigate how many trains should be instantiated in the model and how the length of the trains should be specified for the safety proof to be sound. Additionally, in the implicit model of train movements, the length of a train – in terms of numbers of sections that the train occupies – may vary as the train moves, although the train has a fixed geometric length. This variation reflects the actual view that interlocking systems have on train lengths as they record which sections the trains occupy.

Trains in our model are assumed to be well-behaved, meaning that all of the following hold.

(1) Trains always move according to the actual settings of the physical railway network under consideration, e.g., if the actual position of a point is PLUS (MINUS) then trains can only move from the stem end of the point to the plus (minus) end of the point and vice versa.

(2) Trains do not "fly" – i.e., they do not move from a section t_a to a further section t_b without making a continuous path through the intermediate sections between t_a and t_b.

(3) Trains always stop in front of a signal whose actual aspect is CLOSED, and they only proceed when authorised by a signal whose actual aspect is OPEN.

(4) Trains can change their travel direction, but do not reverse.[7]

(5) Trains do not split.

[7] The difference between *changing direction* and *reversing* is explained further below.

These assumptions are well justified for the following reasons. (1) The first assumption is a physical constraint. (2) The primary causes of "train flying" as seen by interlocking systems are failures in the train detection systems (also known as Track Occupancy Detection systems) which detect whether a section is vacant or occupied by a train [32]. New interlocking systems use axle counters which are far more reliable than track circuits used by legacy systems for train detection. Thus, for simplicity at the high-level design, failures in train detection systems are not included in the model, hence eliminating the possibilities of having "flying trains". (3)-(4) The third and fourth assumptions are taken, as in ETCS Level 2, the sophisticated onboard computer systems in a fitted train ensure that the train would not go further than what it is authorised to move forward and that the train does not accidentally reverse [14, chap. 3]. An emergency brake will be triggered to bring a train to a full-stop if the train violates its movement authority. (5) The fifth assumption is taken as we do not include shunting in our model and for simplicity we assume that trains do not fall apart.

The movements of a train as seen by interlocking systems are like an "elastic band" as illustrated in Figure 8. The band extends to the next section in front of the train when its head starts occupying the next section – movement from (a) to (b) in Figure 8. The band shrinks when the train leaves the section occupied by its tail – movement from (b) to (c) in Figure 8.

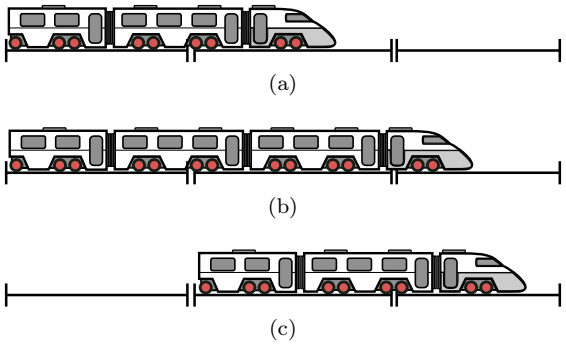


Figure 8: "Elastic band" train movements.

Train movements in the "elastic band" model are categorised into the following types: (1) head movements, (2) tail movements, (3) change direction movements (only on linear sections), (4) entering interlocked area movements, and (5) leaving interlocked area movements. The interlocked area is the network or the fraction of a network under control of the considered interlocking system. The head movements and tail movements respectively model the extensions and shrinking shown in Figure 8. The movements of type (3) model the case where a train changes its travel direction. Note that *change direction* movements differ from *reversing* movements: in a change direction movement, the driver has to move from the front driver cab to the back driver cab and drives the train forward (w.r.t. the direction that the driver cab is facing), while in a reversing movement the driver stays in the front driver cab and drives the train backward [14, 5.12-13]. Change direction movements are allowed in our model of train movements, while reversing movements are not allowed. The movements of type (4) and (5) account for the movements at the boundary of the network under control. In other words, they model how trains are put into the network and taken out of the network under consideration. The subsequent paragraphs describe in details the movements of each type.

Head Movements. A head movement can occur on a section e if it is occupied by a train and the head of the train is in the section, i.e., the H and O bits of the variable representing the occupancy status of e are set (see Section 7.1). If e is a linear section and has a signal s intended for the same direction as the direction of the train, then the actual aspect of s has to be OPEN. The effect of the head movement is illustrated in Figure 9: the head of the train leaves e, effectively toggles the H bit of e's occupancy status variable; and then the head of the train enters the next section in the travel direction – in this case $up(e)$, consequently toggles the H and O bits of the occupancy status variable of $up(e)$. If e is a point, the train will move

according to the physical (actual) position of the point. For example, if the position of the point is plus, then train will move from the plus end to the stem end or vice versa.

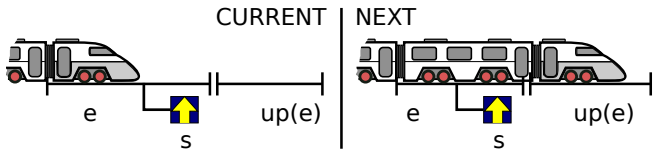


Figure 9: Example of head movement on section e from down to up.

The conditions and effects on occupancy status variables can be modelled efficiently by bit-wise AND (&), OR (|), and XOR ($\oplus$) operators and arithmetic shift left ($\ll$) and arithmetic shift right ($\gg$) operators. The following proposition models the head movement transition in the example shown in Figure 9.

$$(e.D2U \;\&\; 0b101) = 0b101 \wedge s.ACT = OPEN \\ \wedge\, e.D2U' = (e.D2U \oplus 0b100) \wedge up(e).D2U' = (up(e).D2U \oplus 0b101) \quad (19)$$

Tail Movements. A tail movement can occur on a section e if it is occupied by a train and the tail of the train is within the section, while the head of the train is not in e. This means that e's occupancy status variable has the T and O bits set, and the H bit unset, i.e., its value is $0b011$. Figure 10 illustrates the effect a tail movement: the tail of the train vacates e, hence resetting e's occupancy status variable to 0; then the tail of the train moves to the next section in the travel direction – in this case $up(e)$, consequently toggles the T bit of the occupancy status variables of $up(e)$. As with head movements, if e is a point, then tail movements on it will depend on e's physical position.

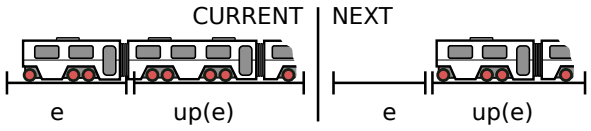


Figure 10: Example of tail movement on section e from down to up.

The following proposition specifies the tail movement transition in the example shown in Figure 10.

$$e.D2U = 0b011 \wedge e.D2U' = 0b000 \wedge up(e).D2U' = (up(e).D2U \oplus 0b010) \quad (20)$$

Change Direction Movements. Change direction movements allow trains to change their travel direction to the opposite. As specified in [14, 5.12-13], a change direction movement has to follow a strict procedure that requires the train to have reached the end of its movement authority and be at a stand-still position. For simplicity, it is assumed that a train is only allowed to change its travel direction on a linear section which has signals intended for both up and down directions as shown in Figure 11. A train can change its direction on a section e when the signal the train is facing has the actual aspect of CLOSED, and the whole train is inside e. These conditions ensure that no further movements can be made forward by the train, i.e., the train is in a stand-still position. The effect of the movement is straight-forward: the values of the occupancy status variables $e.D2U$ and $e.U2D$ are swapped.

The following proposition specifies the change direction movement in the example shown in Figure 11.

$$e.D2U = 0b111 \wedge s2.ACT = CLOSED \wedge e.D2U' = e.U2D \wedge e.U2D' = e.D2U \quad (21)$$

Note that we do not check the setting of *s1* in the condition, however the existence of *s1* is essential, since the interlocking controller should use it to prevent the unauthorised movements of the train after it

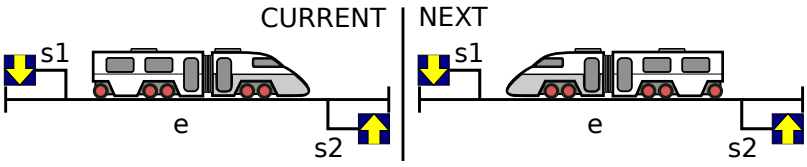


Figure 11: Change direction from up to down. It is analogous from down to up.

has changed the direction.

Entering Interlocked Area Movements. A train enters an interlocked area by two steps: (a) first the head of the train enters the interlocked area, then (b) the tail enters the interlocked area as shown in Figure 12 and Figure 13, respectively. The details about these movements are explained in the following.

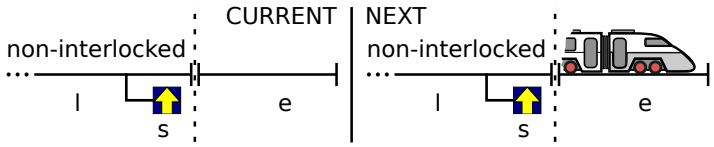


Figure 12: The head of a train enters the interlocked area.

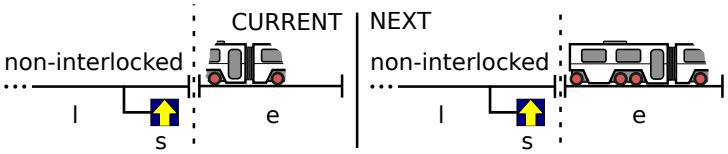


Figure 13: The tail of a train enters the interlocked area.

(a) The head of a train can enter the section e at the boundary of the interlocked area when the actual aspect of signal s is OPEN, as shown in Figure 12. Consequently, the head of the train is simply "put" on e – i.e., the H and O bits of e's occupancy status variable are toggled. The following proposition expresses the transition for the example shown in Figure 12.

$$s.ACT = OPEN \wedge e.D2U' = (e.D2U \oplus 0b101) \tag{22}$$

(b) The tail of the train can enter the boundary section e after the head has entered the section. The condition is that e is occupied by a train without its tail in the section, as shown in the left of Figure 13. If this condition holds, then the tail of the train will be "put" into the network – i.e., the T bit of e occupancy status variable is toggled – as illustrated in the right of Figure 13. The following proposition expresses the transition for the example in Figure 13.

$$(e.D2U \ \& \ 0b011) = 0b001 \wedge e.D2U' = (e.D2U \oplus 0b010) \tag{23}$$

Leaving Interlocked Area Movements. Similar to entering interlocked area movements, a train leaves an interlocked area in two steps: (a) first the head of the train leaves the interlocked area, then (b) the tail leaves the interlocked area as illustrated by Figure 14 and Figure 15, respectively. These movements are further elaborated below.

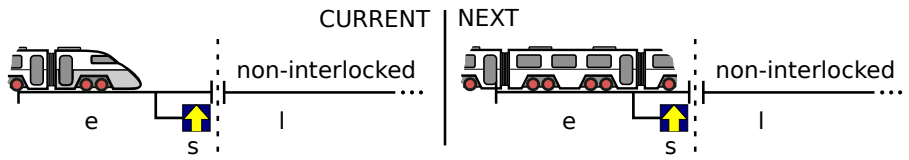


Figure 14: The head of a train leaves the interlocked area.

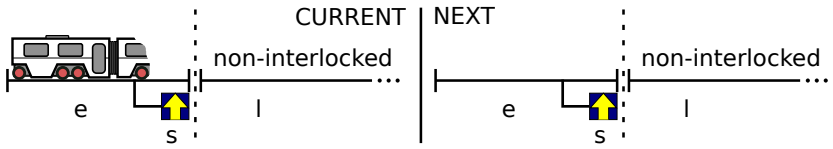


Figure 15: The tail of a train leaves the interlocked area.

(a) The train can leave the interlocked area at a boundary section e if the train occupies e in the direction toward the non-interlocked area and the head of the train is inside e. As the effect, the head of the train is removed from e – i.e., the H bit of e's occupancy status variable is toggled. The following proposition specifies the transition for the movement shown in Figure 14.

$$(e.D2U \ \& \ 0b101) = 0b101 \wedge e.D2U' = (e.D2U \oplus 0b100) \tag{24}$$

(b) The tail of a train can leave the interlocked area at a boundary section e if e is occupied in the direction toward the non-interlocked area, the tail of the train is in the section, and the head of the train is not in e. The train is removed from the interlocked area – i.e., e's occupancy status variable is reset to 0 – as the effect of the transition. The movement in Figure 15 is modelled by the following proposition.

$$e.D2U = 0b011 \wedge e.D2U' = 0b000 \tag{25}$$

Note that in Figure 14, the signal s is not controlled by the considered interlocking system, but the neighbouring one. Therefore, in our model, we do not check the setting of s for the movement shown in Figure 14. Intuitively, this is an over-approximation because trains would move more freely than they are supposed to. This over-approximation does not jeopardise the soundness of the safety proof of the considered interlocking. Moreover, if we want to verify the safety of the combined model of two interlockings, the train movement model can easily be restricted to allow trains to pass s only when s is OPEN. Thus, the safety of the combined model can be inferred from the safety of the two element interlocking models. A formal proof for such composition may be made, however it is out of the scope of this work.

8. High-level Safety Properties Generation

This section outlines the formal safety properties of an interlocking system generated in step (4) of the verification process described in Section 4.

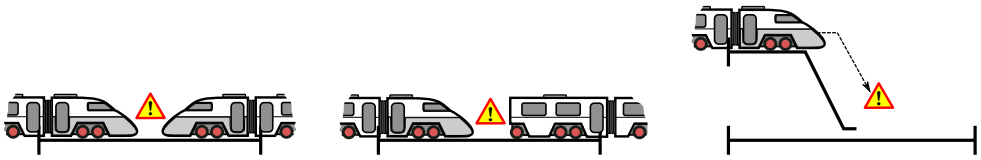
We have developed a *properties generator* that can be used to instantiate generic safety properties (in the form of property patterns) with configuration data specified in the DSL described in Section 5. In step (4) of our method, this generator is applied to the configuration data that was created in step (1) and verified to be statically well-formed in step (2) of our method. This results in concrete properties expressed as state invariants in the Kripke structure generated in step (3) of our method. Below we give more details on this.

Interlocking systems must at least guarantee the high-level safety properties of no-collisions and no-derailments. These properties can be expressed as invariants over the occupancy status variables of linear and point sections in the given network layout. Basically, an interlocking system is safe if no hazardous

situations occur on any linear or point sections at any time. Thus, the high-level safety properties can be expressed formally by the following invariant.

$$\phi = \neg(\bigvee_{l \in \mathcal{L}} \mathcal{H}_l \vee \bigvee_{p \in \mathcal{P}} \mathcal{H}_p) \tag{26}$$

where $\mathcal{L}$ is the set of all linear sections and $\mathcal{P}$ is the set of all points in the network layout of the given configuration data, and $\mathcal{H}_l$ and $\mathcal{H}_p$ specify conditions for hazards to occur on a linear section l and a point p, respectively. These propositions are disjunctions of sub-propositions expressing hazards of different types on a section as shown in Figure 16 such as: (a) head-to-head collision on a section, (b) head-to-tail collision on a section, or (c) derailment on a point. Some examples of sub-propositions are given in the subsequent paragraphs.



(a) Head-to-head collision (b) Head-to-tail collision (c) Derailment on points

Figure 16: Hazardous situations on a section.

Head-to-head collision on a linear section. A head-to-head collision occurs on a linear section l, when two trains running in opposite directions meet in l. This situation is expressed by the following formula where $l.D2U$ ($l.U2D$) is the variable representing the occupancy status of the section in the travel direction from down (up) to up (down).

$$l.D2U * l.U2D > 0 \tag{27}$$

As $l.D2U * l.U2D > 0$ iff $l.D2U > 0$ and $l.U2D > 0$, the formula expresses that the section is occupied in both down-to-up ($l.D2U > 0$) and up-to-down ($l.U2D > 0$) directions. Head-to-head collisions on points are formulated in the similar way.

Head-to-tail collision on a linear section. A head-to-tail collision occurs on a linear section l when a train T_2 enters l while it is already occupied by another train T_1 travelling in the same direction. Although two trains may never collide if l is long enough to accommodate both of them, or if T_1 travels at higher speed than T_2 does, these cases are still considered as collisions. These situations are reflected by the values of the occupancy status variables of l. For example, when l is occupied by a train T_1 in the direction from down to up, $l.D2U$ will have its O bit set. If another train T_2 enters l in the same direction, the H and O bits of $l.D2U$ will be toggled according to the train movement model described in Section 7.7, resulting in $l.D2U$ having O bit unset, while at least one of its H or T bits are set. Therefore, a head-to-tail collision on l in the direction up is detected by the following formula where & is *bit-wise and* operator.

$$l.D2U * (1 - (l.D2U \ \& \ 1)) > 0 \tag{28}$$

As Formula 28 holds iff $l.D2U > 0$ and $1 - (l.D2U \ \& \ 1) > 0$, the formula expresses that the section is occupied in direction up ($l.D2U > 0$) while the O bit of l is unset ($1 - (l.D2U \ \& \ 1) > 0$). Head-to-tail collisions on linear sections in the down direction and on points are formulated analogously.

Derailment on a point. A derailment occurs when a train traverses a point p which is not locked in the correct position for the travel direction of the train. This situation is expressed by the following formula where $p.POS$ is the point's actual position, $p.S2PM$, $p.P2S$, and $p.M2S$ are variables representing the

occupancy status of the point in the travel direction entering the point from its stem, plus, or minus ends, respectively, and $\gg$ is arithmetic shift right operator.

$$p.POS * p.P2S + (1 - (p.POS \ \& \ 1)) * p.M2S + (p.POS \gg 1) * p.S2PM > 0 \tag{29}$$

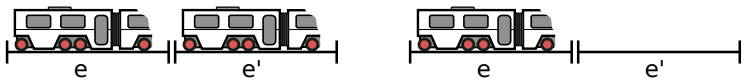
Formula 29 captures the following cases: (a) a train is entering a point from its plus end ($p.P2S > 0$) while the point is in not in the plus position ($p.POS > 0$); (b) a train is entering a point from its minus end ($p.M2S > 0$) while the point is not in the minus position ($1 - (p.POS \ \& \ 1) > 0$); and (c) a train is entering a point from its stem end ($p.S2PM > 0$) while the point is in the intermediate position ($(p.POS \gg 1) > 0$).

9. Verification of High-level Safety Properties

This section explains step (5) of the modelling and verification process described in Section 4.

In this step the task is to verify that the the safety property ϕ (see Formula 26) generated in step (4) is an invariant in the Kripke structure K generated in step (3). To verify that, we use a bounded model checker (RT-Tester) to perform k-induction as explained in Section 2. As mentioned in Section 2, we strengthen ϕ with another invariant ψ in order to eliminate spurious counterexamples. Then instead of proving that ϕ is an invariant in K, we prove that $\phi \wedge \psi$ is an invariant in K. ψ is a conjunction of a number of propositions constraining the starting state of the induction step. An example of such propositions is given in the following.

Train Integrity. Some states are not feasible and are not reachable in the model of a given interlocking system because the combinations of the values of the occupancy status variables of sections (see Section 7) in such states reflect situations that are not physically possible. Examples of infeasible states are shown in Figure 17. A section e is occupied by a train *T1*, and the head of the train is not in e; while the next section e' of e in the travel direction is occupied by another train *T2* as shown in Figure 17a, or e' is vacant as shown in Figure 17b. These situations are physically impossible, because they imply that the train *T1* does not have a head.



(a) (b)

Figure 17: Examples of infeasible states ×

In order to eliminate such states, we strengthen the safety properties with *train integrity* invariants. As illustrated intuitively in Figure 18, train integrity invariants, as the name suggests, ensure the integrity of all the trains (modelled implicitly by the occupancy status variables of sections) in the model, i.e., every train is a unity object with a head and a tail, except for trains that are entering or leaving the considered network. The idea is simple: *starting from a section e that is occupied by the head (tail) of a train, if we search backward (forward) – w.r.t. the train's travel direction – we should find a unified train without a gap.* In other words, we should eventually find (1) the tail (head) of the same train somewhere in one of the previous (next) sections of e – w.r.t. to the train's direction – or (2) the boundary of the interlocked area, before we find the head (tail) of another train or a vacant section.

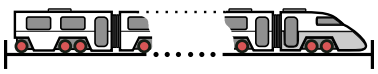


Figure 18: Train integrity invariant illustration ✓

The train integrity invariants can be formalised as a conjunction of formulas over the track occupancy variables. For each possible travel direction on a non-boundary section e, there is a formula expressing the

constraints between the occupancy status variable of e and the occupancy status variable of the next section e' in the same travel direction. The pattern of such a formula depends on which type of section (linear or point) the current section e and the next section e' are. For instance, for travel direction *up* and a linear section e that has another linear section e' as the neighbour in travel direction *up* – i.e., $e' = up(e)$, the formula will take the following form:

$$(e.D2U \ \& \ 0b101) = 0b001 \iff (e'.D2U \ \& \ 0b011) = 0b001 \tag{30}$$

This formula expresses that section e is occupied by a train in direction *up* (the O bit of $e.D2U$ is 1) without the head of the train being on the section (the H bit of $e.D2U$ is 0), iff section e' is also occupied by a train in direction *up* (the O bit of $e'.D2U$ is 1) without the tail of the train being on the section (the T bit of $s'.D2U$ is 0). It can easily be seen that Formula 30 rules out the situations shown in Figure 17. Formula 30 shows the expressiveness of our state encodings allowing properties to be efficiently and compactly formulated.

10. Experiments

This section first presents some verification experiments with our toolchain, and then it presents some benchmarking results in comparison with different invariant checking techniques supported by nuXmv.

10.1. Experiments with Our Toolchain

We have used the toolchain described in Section 4 to generate safety properties and model instances for a number of railway networks (cases) as shown in Table 2 and to verify that the generated safety properties hold for the generated model instances. The cases in Table 2 are listed in the approximate order of increasing network complexity. The first seven cases are made-up networks inspired by the typical examples used in other studies about formal verification of railway interlocking systems [20, 38, 25, 24]. For instance, the example network shown in Figure 1 is case *Mini* in Table 2. Gadstrup-Havdrup (Gt-Hd) and Køge are real networks extracted from the Early Deployment Line (EDL) in the Danish Signalling Programme. The EDL is the first regional line in Denmark to be commissioned in the Danish Signalling Programme. The line spreads over 55 kilometres from Roskilde station to Næstved station. There are in total eight stations in the EDL ranging from simple stations similar to the one shown in Figure 1, to complex stations such as Køge. The network descriptions (in XML representation) and the corresponding generated properties and model instances for the first seven cases can be found at `http://www.imm.dtu.dk/~aeha/RobustRailS/data/casestudy`.

Table 2: Verification metrics for different networks when using simple induction ($k = 1$).

	Linears	Points	Signals	Routes	$log_{10}(\|S\|)$	Time	Memory
Tiny	3	0	4	2	11	1	22
Toy	6	1	6	4	26	3	87
Twist	8	2	8	8	39	10	177
Fork	9	2	8	6	40	10	171
Cross	8	2	8	10	41	15	192
Mini	6	2	8	12	37	17	199
Lyngby	11	6	14	24	79	243	867
Gt-Hd	21	5	24	33	113	230	1159
Køge	57	23	60	73	332	5415	8436
EDL	110	39	126	179	651	19678	23413

In our first trials of verifying the models, we used simple induction (k-induction with $k = 1$) with only safety properties. As discussed in Section 9, we got spurious counterexamples because the safety properties

are not strong enough to be inductive. In order to remedy the issues, we tried two different approaches: (1) increasing k, and (2) strengthening the invariant to be verified (in this case, the safety properties). It turned out that the verification time increased significantly as k increased in the former approach, making it impractical to verify the properties even for the small networks. In the latter approach, we were able to derive strengthening properties ψ (see Section 9) for which the verification could be done just using simple induction. According to De Moura et al. [27], any k-induction proof can be reduced to a simple induction proof with invariant strengthening. Note, however, that in some applications k-induction has been shown to be advantageous, see, for example, [23]. Table 2 shows metrics for the final verification using the invariant strengthening approach. Each row of the table lists the size of a network in terms of the number of linear sections, points, signals, and routes in the configuration, and the approximate number of possible states ($|S|$) in the corresponding model instance. For brevity, the approximate number of states are represented in their common logarithm values ($log_{10}(|S|)$); for example, the *tiny* case has approximately 10^{14} possible states. The two last columns show the approximate accumulated verification time (in seconds) and memory usage (in MB). All experiments have been performed on a machine with Intel(R) Xeon(R) CPU E5-2667 0 @ 2.90GHz, 64GB RAM, CentOS 6.6, Linux 2.6.32-504.8.1.el6.x86_64 kernel.

We also injected errors into models. Counterexamples for these were normally found in relatively short time. This appears to be a general trend when dealing with interlocking systems [26]. In a few cases, it took long time to find counterexamples. Counterexamples show how errors in the model or the configuration data lead to violations of properties, which may be overlooked by manual inspection. For example, if a check for the status of protecting points was missing while allocating a route, it would result in a counterexample where a point is commanded to move while it is in use.

10.2. Comparison with Other Techniques

In order to study how other invariant checking techniques perform on our models compared to our toolchain, we translated our models into the input language of nuXmv. nuXmv is "a new symbolic model checker for the analysis of synchronous finite-state and infinite-state systems" [9]. It extends one of the most popular open-source model checkers, NuSMV2 [10], with support for various state-of-the-art abstraction techniques such as Binary Decision Diagram (BDD)-based techniques [8], CounterExample-Guided Abstraction Refinement (CEGAR) [11], Incremental Construction of Inductive Clauses for Indubitable Correctness (IC3) [7], and Cone of Influence (COI) [4] techniques. The subsequent paragraphs report the translation, techniques, configurations, settings, results, and discussion of our benchmarking experiments.

Translation. In order to ensure a faithful translation as the ground for our comparison, the following principles were used for translating the behavioural models and properties to the input language [6] of nuXmv.

- Variable declarations representing the state space of a Kripke structure model were translated to corresponding variable declarations in nuXmv using the type `unsigned word` [6].
- The transition relation of a Kripke structure model was translated to a `TRANS` constraint [6] in nuXmv.
- Safety properties and strengthening invariants were translated to nuXmv's invariant specifications `INVARSPEC` [6].

Techniques. We tried to verify the safety properties in the nuXmv translation of the concrete models using various invariant checking techniques [9] supported by nuXmv as listed in the following.

(1) `check_invar`: BDD-based invariant checking using reachability analysis.

(2) `check_invar_bmc_inc`: induction using a SAT solver, incremental algorithm.

(3) `check_invar_cegar_predabs`: performs CEGAR [11] loop.

(4) `check_invar_ic3`: checking invariant using the IC3 [7] engine in which instead of unrolling the transition relation, the given property is gradually refined using clauses that are inductive relative to stepwise approximate reachability.

(5) `check_invar_inc_coi_bdd`: BDD-based incremental COI [4] invariant checking.

(6) `msat_check_invar_inc_coi`: SMT-based incremental COI invariant checking.

Configurations. The techniques are run on three different configurations of the properties to be verified denoted by the suffixes appended to the name of the technique as described in the following.

(a) `all`: the technique is used to verify the conjunction of all properties including both safety properties and strengthening invariants.

(b) `safety`: the technique is used to verify the conjunction of all safety properties (strengthening invariants are not included).

(c) `safety_individually`: the technique is used to verify each safety property individually (strengthening invariants are not included).

For example, `check_invar_all` means that `check_invar` is used to verify the conjunction of all properties, including both safety properties and strengthening invariants.

These configurations allow us to study whether other techniques can verify the safety properties efficiently without strengthening invariants, which are required for a successful verification in our toolchain. Note that some techniques work only with a subset of the above configurations (`all`). In such cases, only results of working configurations are reported.

Settings. All experiments are run with nuXmv 1.0.0 on the same machine specified in Section 10.1, where the evaluation with our toolchain in RT-Tester has been performed. Each technique is run on networks with increasing complexity as listed in Table 2. Running time and memory usage are profiled during the experiments. *Running time threshold* is 24 hours, and *memory usage threshold* is 54G. These thresholds are chosen based on the following factors: the computation capacity of the machine where the experiments were run, the verification results of our toolchain as shown in Table 2, and the limit amount of time we had for experiments (experiments should not be run forever). If an experiment exceeds at least one of these two thresholds and produces no conclusion so far, it will be *terminated*. Once an experiment is terminated, no further experiments will be run for the same technique because the technique would simply reach the thresholds again for more complex networks.

Results. Table 3 shows the running time and memory usage collected from the benchmarking experiments where `rt_tester` denotes our toolchain implemented in RT-Tester. For each technique, the running time (in seconds) of *successful* experiments are listed in the first row (with white background and underlined), while the memory usage (in MB) of successful experiments are listed in the second row (with white background). Experiments that were *terminated* are marked with *grey* background denoting that they have exceeded the running time threshold or the memory threshold without producing any conclusion. The value "-" denotes the experiments that were not run because the experiments with the same technique on smaller cases have been terminated due to too long running time or memory exhaustion. Few techniques, e.g., `check_invar_bmc_inc`, have running time and memory usage of zero for `tiny` case because the experiments ran too quick for the profiler to measure.

Discussion. As seen in Table 3, among all of the techniques used in the benchmarking experiments, only our toolchain (`rt_tester`) was able to deliver a conclusion for the largest case, the EDL, within the running time and memory limits. All other techniques exceeded either the running time threshold or the memory threshold without delivering a conclusion for the EDL case, hence they were terminated.

Table 3: Benchmarking results – verification time and memory usage

$\frac{n}{m}$	verification succeeded, took n seconds and used m MB of memory	$\frac{n^*}{}$	running time threshold exceeded, terminated, no conclusion	$m^\dagger$	memory threshold exceeded, terminated, no conclusion	-	canceled because its previous one has been terminated		

	Tiny	Toy	Twist	Fork	Cross	Mini	Lyngby	Gt-Hd	Køge	EDL
`rt_tester_all`	$\frac{1}{22}$	$\frac{3}{87}$	$\frac{10}{177}$	$\frac{10}{171}$	$\frac{15}{192}$	$\frac{17}{199}$	$\frac{243}{867}$	$\frac{230}{1159}$	$\frac{5415}{8436}$	$\frac{19678}{23413}$
(1a) `check_invar_all`	$\frac{1}{39}$	$\frac{9}{285}$	$\frac{86401^*}{}$	-	-	-	-	-	-	-
				-	-	-	-	-	-	-
(1b) `check_invar_safety`	$\frac{1}{0}$	$\frac{9}{254}$	$\frac{86401^*}{}$	-	-	-	-	-	-	-
				-	-	-	-	-	-	-
(1c) `check_invar_safety_individually`	$\frac{2}{0}$	$\frac{8}{257}$	$\frac{86401^*}{}$	-	-	-	-	-	-	-
				-	-	-	-	-	-	-
(2a) `check_invar_bmc_inc_all`	$\frac{2}{0}$	$\frac{1}{60}$	$\frac{2}{113}$	$\frac{2}{107}$	$\frac{3}{121}$	$\frac{2}{113}$	$\frac{15}{487}$	$\frac{13}{974}$	$\frac{148}{15148}$	$55318^\dagger$
(3a) `check_invar_cegar_predabs_all`	$55297^\dagger$	-	-	-	-	-	-	-	-	-
		-	-	-	-	-	-	-	-	-
(4a) `check_invar_ic3_all`	$\frac{1}{54}$	$\frac{2}{116}$	$\frac{7}{227}$	$\frac{4}{217}$	$\frac{7}{244}$	$\frac{10}{227}$	$\frac{146}{993}$	$\frac{1783}{2712}$	$\frac{86401^*}{}$	-
										-
(4b) `check_invar_ic3_safety`	$\frac{2}{52}$	$\frac{26}{116}$	$\frac{233}{539}$	$\frac{190}{210}$	$\frac{555}{233}$	$\frac{1260}{239}$	$\frac{86401^*}{}$	-	-	-
								-	-	-
(4c) `check_invar_ic3_safety_individually`	$\frac{2}{52}$	$\frac{64}{107}$	$55340^\dagger$	-	-	-	-	-	-	-
				-	-	-	-	-	-	-
(5a) `check_invar_inc_coi_bdd_all`	$\frac{1}{0}$	$\frac{9}{289}$	$\frac{86400^*}{}$	-	-	-	-	-	-	-
				-	-	-	-	-	-	-
(5b) `check_invar_inc_coi_bdd_safety`	$\frac{1}{0}$	$\frac{9}{269}$	$\frac{86401^*}{}$	-	-	-	-	-	-	-
				-	-	-	-	-	-	-
(5c) `check_invar_inc_coi_bdd_safety_individually`	$\frac{1}{0}$	$\frac{52}{347}$	$\frac{86401^*}{}$	-	-	-	-	-	-	-
				-	-	-	-	-	-	-
(6a) `msat_check_invar_inc_coi_all`	$\frac{2}{0}$	$\frac{2}{69}$	$\frac{4}{125}$	$\frac{4}{117}$	$\frac{4}{133}$	$\frac{6}{134}$	$\frac{98}{609}$	$\frac{111}{1172}$	$\frac{26035}{18946}$	$55305^\dagger$

As discussed in Section 2.2, checking techniques based on complete model representations also failed for the nuXmv tool. The BDD-based invariant checking technique – `check_invar` – did not perform well on our models. It exceeded the running time threshold for small cases on all of the three configurations.

BDD-based incremental COI technique – `check_invar_inc_coi_bdd` – did not perform better compared to its BDD-based peer – `check_invar`. It exceeded the running time threshold for small cases on all three configurations.

CEGAR technique – `check_invar_cegar_predabs` – did not perform well on our models either. When run with configuration `all`, it exceeded the memory threshold while refining the abstraction, even for the smallest case. When run with configuration `safety`, or configuration `safety_individually`, it gave a segmentation fault error (thus the results are not shown in Table 3).

IC3 technique – `check_invar_ic3` – was not able to handle the large cases: it exceeded the memory threshold when run on Køge with `all` configuration, and exceeded the running time threshold on even smaller cases when run with `safety` or `safety_individually` configurations.

Induction using an SMT solver – `msat_check_invar_inc_coi` – performed better than our toolchain for small cases. For large cases, e.g., Køge, it used more memory and time than our toolchain. Consequently, it exceeded the memory threshold without delivering a conclusion for the EDL case.

It is evident that induction using a SAT solver in nuXmv – `check_invar_bmc_inc` – outperformed our toolchain for cases up to Køge as far as the running time is concerned. On the other hand, it used increasingly more memory compared to our toolchain as the size of cases grows. As a consequence, the technique exceeded the memory threshold without delivering a conclusion for the EDL case. Based on the rate of increasing memory usage of `check_invar_bmc_inc` and our toolchain, we would expect the technique to use – for the EDL case – at least six times the amount of memory it used for verifying Køge case, i.e., at least 90GB. Thus our memory threshold is not the limiting factor preventing the technique from succeeding on the EDL case. An analogous argument holds for `msat_check_invar_inc_coi`.

The better running time of `check_invar_bmc_inc_all` can be explained by the fact that the BMC algorithms implemented in RT-Tester are optimised for situations where witnesses *can* be found, because these witnesses typically represent test cases. In the model verification cases considered here, the verification goal is always to show that witnesses *cannot* be found. Moreover, the inductive techniques used by `check_invar_bmc_inc_all` are based on temporal induction according to [12]; this approach shows some subtle optimisations when compared to our k-induction strategy which is based on [30]. In RT-Tester, the induction strategy is implemented in a separate layer on top of the SMT solver. This layer could be exchanged by another one also implementing the strategies described in [12].

11. Related Work

In recent years, the railway domain has become one of the most promising application domains of formal methods. Several research groups have investigated how formal methods would help efficiently producing more robust railway control systems. An overview of recent trends can be found in [16], and recommendations and best-practices for efficient development and verification of safe railway control systems are summarised in [22]. Re-configurable systems and automated verification are among these recommendations that we have followed.

Model checking is a promising technique for verifying safety properties of interlocking systems thanks to its capability to be fully automated. Unfortunately, due to the state space explosion problem, the technique is only able to verify applications of small size [17]. Several techniques have been proposed in order to push the applicability bounds toward industrial size. Winter et al. suggest using ordering strategies optimised for interlocking models [38]. A number of high-level abstractions for reducing the complexity of interlocking models are presented in [25]. In [15], Fantechi et al. suggest a distributed interlocking model whose verification can be divided into small tasks and verified in parallel. SAT-based model checking and slicing technique are used in [26]. In order to remedy the problem with state space explosion in the global model checking approach, we have recently used BMC for the work described here and for some other applications [23, 24]. In the current work, a combination of SMT-based BMC with inductive reasoning

allowed us to verify safety properties without having to explore the whole state space, hence we were able to push the bounds even further to handle larger networks of industrial size. As an alternative to the model checking approach, theorem-proving-based techniques have also shown success in the railway domain, see, e.g., [2, 21], but these provide a lesser degree of automation.

Although sequential release has been used in some interlocking systems, we have not found any published formal models of interlocking systems that integrate this feature. In [33], the conditions for elements to be unlocked and reused in sequential releases are pre-computed and specified in the interlocking tables. In our approach, sequential release is integrated into the behavioural model rather than into the configuration data. This reduces the complexity of the configuration data and makes interlocking configuration data relatively independent from the chosen interlocking approaches.

12. Conclusion and Future Work

This article presented a fully automated, formal method and an associated tool suite for verifying the forthcoming Danish railway interlocking systems. The new systems are compatible with the ETCS Level 2 and feature sequential release. A formal model for these systems was outlined. A novelty in our contribution is that the system is part of a signalling system based on ETCS Level 2 for which there are no physical signals along the tracks; instead, movement authorities are communicated to onboard computers via a radio block center (RBC). By introducing the concept of virtual signals, ETCS Level 2 compatible interlocking systems can be formalised in a similar way as conventional interlocking systems with physical signals. Another novelty is that the formal model features sequential release. As a consequence, the model is more complex than those supporting route-based release only, because additional variables and transitions are required. Therefore the verification becomes more challenging. In spite of this difficulty, using a combination of SMT-based BMC and inductive reasoning, we were able to successfully verify safety properties for systems controlling large networks of realistic size. This was enabled by encodings of the state space, the transition relation, and of the safety properties that can be efficiently evaluated by SMT solvers supporting bit vectors and integer arithmetic.

For future work, we will benchmark how sequential release affects the complexity, and hence the verification challenges, of interlocking models. Furthermore, we will investigate advanced techniques for automating the process of discovering strengthening invariants, and for reducing the size of the networks that need to be modelled. For the current model there are potential overlaps between the strengthening invariants, which should be eliminated in order to reduce the size of the formula to be solved by the SMT solver.

The encoding of the transition relation suggested in this article relies on the availability of an SMT solver which can handle bit vector and integer arithmetic. This design decision was motivated by the fact that changes in the model state usually affect several bits, and the bit vector or integer expressions allowed to model this in an intuitive way. It is obvious, however, that a pure Boolean encoding would still be possible to represent the transition relation with acceptable complexity. We plan to investigate whether the resulting SAT-based approach yields any performance gains.

Acknowledgments. The authors would like to thank Ross Edwin Gammon and Nikhil Mohan Pande from Banedanmark (Railnet Denmark) and Jan Bertelsen from Thales for helping us with their expertise about Danish interlocking systems and always being helpful when we had questions; Dr.-Ing. Uwe Schulze and Florian Lapschies from the University of Bremen for their help with the implementation in the RT-Tester toolchain; and the anonymous referees and Alessandro Fantechi for valuable proposals for improving the presentation.

References

[1] M. Aanæs, H.P. Thai, Modelling and Verification of Relay Interlocking Systems, Master's thesis, Technical University of Denmark, DTU Informatics, 2012. Series: IMM-MSC-2012-14.

[2] S. Behnia, A. Mammar, J.M. Mota, N. Breton, P. Caspi, P. Raymond, Industrialising a Proof-based Verification Approach of Computerised Interlocking Systems, in: J. Allan, E. Arias, C. Brebbia, C. Goodman, A. Rumsey, G. Sciutto, N. Tomii (Eds.), 11th International Conference on Computer System Design and Operation in the Railway and Other Transit Systems (COMPRAIL08), WIT Press, 2008.

[3] A. Biere, A. Cimatti, E.M. Clarke, Y. Zhu, Symbolic Model Checking without BDDs, in: R. Cleaveland (Ed.), 5th International Conference on Tools and Algorithms for Construction and Analysis of Systems, TACAS '99, Held as Part of the European Joint Conferences on the Theory and Practice of Software, ETAPS'99, Amsterdam, The Netherlands, March 22-28, 1999, Proceedings, volume 1579 of *Lecture Notes in Computer Science*, Springer, 1999, pp. 193–207.

[4] A. Biere, E.M. Clarke, R. Raimi, Y. Zhu, Verifying Safety Properties of a Power PC Microprocessor Using Symbolic Model Checking without BDDs, in: N. Halbwachs, D. Peled (Eds.), 11th International Conference on Computer Aided Verification, CAV '99, Trento, Italy, July 6-10, 1999, Proceedings, volume 1633 of *Lecture Notes in Computer Science*, Springer, 1999, pp. 60–71.

[5] A. Biere, K. Heljanko, T. Junttila, T. Latvala, V. Schuppan, Linear Encodings of Bounded LTL Model Checking, Logical Methods in Computer Science 2 (2006). arXiv: cs/0611029.

[6] M. Bozzano, R. Cavada, A. Cimatti, M. Dorigatti, A. Griggio, A. Mariotti, A. Micheli, S. Mover, M. Roveri, S. Tonetta, nuXmv 1.0 User Manual, Technical Report, FBK - Via Sommarive 18, 38055 Povo (Trento) - Italy, 2014. Available at https://nuxmv.fbk.eu.

[7] A.R. Bradley, SAT-Based Model Checking without Unrolling, in: R. Jhala, D.A. Schmidt (Eds.), Verification, Model Checking, and Abstract Interpretation - 12th International Conference, VMCAI 2011, Austin, TX, USA, January 23-25, 2011. Proceedings, volume 6538 of *Lecture Notes in Computer Science*, Springer, 2011, pp. 70–87.

[8] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, L.J. Hwang, Symbolic Model Checking: 10^{20} States and Beyond, Inf. Comput. 98 (1992) 142–170.

[9] R. Cavada, A. Cimatti, M. Dorigatti, A. Griggio, A. Mariotti, A. Micheli, S. Mover, M. Roveri, S. Tonetta, The nuXmv Symbolic Model Checker, in: A. Biere, R. Bloem (Eds.), 26th International Conference on Computer Aided Verification, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings, volume 8559 of *Lecture Notes in Computer Science*, Springer, 2014, pp. 334–342.

[10] A. Cimatti, E.M. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, A. Tacchella, NuSMV 2: An OpenSource Tool for Symbolic Model Checking, in: E. Brinksma, K.G. Larsen (Eds.), 14th International Conference on Computer Aided Verification, CAV 2002,Copenhagen, Denmark, July 27-31, 2002, Proceedings, volume 2404 of *Lecture Notes in Computer Science*, Springer, 2002, pp. 359–364.

[11] E.M. Clarke, O. Grumberg, S. Jha, Y. Lu, H. Veith, Counterexample-guided Abstraction Refinement for Symbolic Model Checking, J. ACM 50 (2003) 752–794.

[12] N. Eén, N. Sörensson, Temporal Induction by Incremental SAT Solving, Electronic Notes in Theoretical Computer Science 89 (2003) 543–560.

[13] European Railway Agency, Annex A for ETCS Baseline 3 and GSM-R Baseline 0, 2012. Available at http://www.era.europa.eu/Document-Register/Pages/New-Annex-A-for-ETCS-Baseline-3-and-GSM-R-Baseline-0.aspx.

[14] European Railway Agency, ERTMS – System Requirements Specification – UNISIG SUBSET-026, 2014. Available at http://www.era.europa.eu/Document-Register/Pages/Set-2-System-Requirements-Specification.aspx.

[15] A. Fantechi, Distributing the Challenge of Model Checking Interlocking Control Tables, in: T. Margaria, B. Steffen (Eds.), Leveraging Applications of Formal Methods, Verification and Validation. Applications and Case Studies, volume 7610 of *Lecture Notes in Computer Science*, Springer, 2012, pp. 276–289.

[16] A. Fantechi, Twenty-Five Years of Formal Methods and Railways: What Next?, in: S. Counsell, M. Núñez (Eds.), Software Engineering and Formal Methods, volume 8368 of *Lecture Notes in Computer Science*, Springer, 2014, pp. 167–183.

[17] A. Ferrari, G. Magnani, D. Grasso, A. Fantechi, Model Checking Interlocking Control Tables, in: E. Schnieder, G. Tarnai (Eds.), FORMS/FORMAT 2010 – Formal Methods for Automation and Safety in Railway and Automotive Systems, Springer, 2010, pp. 107–115.

[18] A. Foldager, A Graphical Domain-specific Language for Railway Interlocking Systems, Master's thesis, Technical University of Denmark, DTU Compute, 2015.

[19] H.H. Hansen, J. Ketema, B. Luttik, M.R. Mousavi, J. van de Pol, O.M. dos Santos, Automated Verification of Executable UML Models, in: B.K. Aichernig, F.S. de Boer, M.M. Bonsangue (Eds.), FMCO, volume 6957 of *Lecture Notes in Computer Science*, Springer, 2010, pp. 225–250.

[20] A.E. Haxthausen, M.L. Bliguet, A.A. Kjær, Modelling and Verification of Relay Interlocking Systems, in: C. Choppy, O. Sokolsky (Eds.), 15th Monterey Workshop: Foundations of Computer Software, Future Trends and Techniques for Development, volume 6028 of *Lecture Notes in Computer Science*, Springer, 2010, pp. 141–153.

[21] A.E. Haxthausen, J. Peleska, Formal Development and Verification of a Distributed Railway Control Systems, in: IEEE Transactions on Software Engineering, volume 26, IEEE, 2000, pp. 687–701.

[22] A.E. Haxthausen, J. Peleska, Efficient Development and Verification of Safe Railway Control Software, in: Railways: Types, Design and Safety Issues, Nova Science Publishers, Inc., 2013, pp. 127–148.

[23] A.E. Haxthausen, J. Peleska, S. Kinder, A Formal Approach for the Construction and Verification of Railway Control Systems, in: Formal Aspects of Computing, volume 23, Springer, 2011, pp. 191–219.

[24] A.E. Haxthausen, J. Peleska, R. Pinger, Applied Bounded Model Checking for Interlocking System Designs, in: S. Counsell, M. Núñez (Eds.), Software Engineering and Formal Methods, volume 8368 of *Lecture Notes in Computer Science*, Springer, 2014, pp. 205–220.

[25] P. James, F. Möller, H.N. Nguyen, M. Roggenbach, S. Schneider, H. Treharne, M. Trumble, D. Williams, Verification of Scheme Plans Using CSP||B, in: S. Counsell, M. Núñez (Eds.), Software Engineering and Formal Methods, volume 8368

of *Lecture Notes in Computer Science*, Springer, 2014, pp. 189–204.

[26] P. James, M. Roggenbach, Automatically Verifying Railway Interlockings Using SAT-based Model Checking, in: Electronic Communications of the EASST, volume 35, EASST, 2011.

[27] L.M. de Moura, H. Rueß, M. Sorea, Bounded Model Checking and Induction: From Refutation to Verification, in: W.A.H. Jr., F. Somenzi (Eds.), 15th International Conference on Computer Aided Verification, CAV 2003, Boulder, CO, USA, July 8-12, 2003, Proceedings, volume 2725 of *Lecture Notes in Computer Science*, Springer, 2003, pp. 14–26.

[28] J. Peleska, Industrial-Strength Model-Based Testing - State of the Art and Current Challenges, in: A.K. Petrenko, H. Schlingloff (Eds.), Proceedings 8th Workshop on Model-Based Testing, Rome, Italy, volume 111 of *Electronic Proceedings in Theoretical Computer Science*, Open Publishing Association, 2013, pp. 3–28.

[29] J. Peleska, E. Vorobev, F. Lapschies, Automated Test Case Generation with SMT-Solving and Abstract Interpretation, in: M. Bobaru, K. Havelund, G. Holzmann, Joshi (Eds.), NASA Formal Methods, volume 6617 of *Lecture Notes in Computer Science*, Springer, 2011, pp. 298–312.

[30] M. Sheeran, S. Singh, G. Stålmarck, Checking Safety Properties Using Induction and a SAT-Solver, in: W.A.H. Jr., S.D. Johnson (Eds.), 3rd International Conference on Formal Methods in Computer-Aided Design, FMCAD 2000, Austin, Texas, USA, November 1-3, 2000, Proceedings, volume 1954 of *Lecture Notes in Computer Science*, Springer, 2000, pp. 108–125.

[31] C. European Committee for Electrotechnical Standardization, EN 50128:2011 – Railway applications – Communications, signalling and processing systems – Software for railway control and protection systems, 2011.

[32] G. Theeg, S.V. Vlasenko, E. Anders, Railway Signalling & Interlocking: International Compendium, Eurailpress, 2009.

[33] D. Tombs, N. Robinson, G. Nikandros, Signalling Control Table Generation and Verification, in: CORE 2002: Cost Efficient Railways through Engineering, Railway Technical Society of Australasia/Rail Track Association of Australia, 2002, p. 415.

[34] Verified Systems International GmbH, RT-Tester Model-Based Test Case and Test Data Generator - RTT-MBT - User Manual, 2013. Available on request from http://www.verified.de.

[35] L.H. Vu, Formal Development and Verification of Railway Control Systems - In the context of ERTMS/ETCS Level 2, Ph.D. thesis, Technical University of Denmark, DTU Compute, 2015.

[36] L.H. Vu, A.E. Haxthausen, J. Peleska, A Domain-Specific Language for Railway Interlocking Systems, in: E. Schnieder, G. Tarnai (Eds.), FORMS/FORMAT 2014 - 10th Symposium on Formal Methods for Automation and Safety in Railway and Automotive Systems, Institute for Traffic Safety and Automation Engineering, Technische Universität Braunschweig, 2014, pp. 200–209.

[37] L.H. Vu, A.E. Haxthausen, J. Peleska, Formal Modeling and Verification of Interlocking Systems Featuring Sequential Release, in: C. Artho, P.C. Ölveczky (Eds.), Formal Techniques for Safety-Critical Systems, volume 476 of *Communications in Computer and Information Science*, Springer, 2015, pp. 223–238.

[38] K. Winter, Optimising Ordering Strategies for Symbolic Model Checking of Railway Interlockings, in: T. Margaria, B. Steffen (Eds.), Leveraging Applications of Formal Methods, Verification and Validation. Applications and Case Studies, volume 7610 of *Lecture Notes in Computer Science*, Springer, 2012, pp. 246–260.